

CRUD en MongoDB

MongoDB es un sistema gestor de bases de datos documentales NoSQL, habitualmente nos referiremos a ella con simplemente "Mongo".

Mongo surge en el 2009, en pleno boom de del Big Data, y enseguida alcanza un éxito que se mantiene hasta hoy en día. Su naturaleza escalable, característica fundamental de las bases de datos NoSQL, y su facilidad para realizar búsquedas, característica fundamental de las bases de datos relacionales, hacen que esta combinación sea ideal para introducirnos en el mundo NoSQL.

De las bases de datos NoSQL hablaremos en el tema 3 que está dedicado exclusivamente a ellas. Por ahora será suficiente con decir que las características fundamentales de las bases de datos NoSQL son lidiar con datos no estructurados y su capacidad para crecer horizontalmente.

Las bases de datos relacionales como MySQL, PostgreSQL, SQLServer... tienen un esquema mediante el que se definen las tablas y sus relaciones. En las asignaturas de bases de datos aprendemos a diseñar este esquema y normalizar tablas de manera que eliminemos la información redundante. Cada tabla está compuesta por registros con exactamente la misma estructura. Una base de datos bien diseñada tiende a tener información en todos los campos de todos los registros. Si nuestra tabla tiene muchas celdas en blanco se considera que está mal diseñada.

En el sistema gestor de bases de datos MongoDB se pueden crear bases de datos independientes. Dentro de estas bases de datos tenemos colecciones, un concepto similar a las tablas relacionales. Dentro de las colecciones se guardan documentos como si fueran los registros de las bases de datos relacionales. Esta es la clave de MongoDB.

Los documentos se almacenan internamente en forma de BSON, el formato binario de JSON. Para nosotros será totalmente transparente y trabajaremos con los documentos como si fueran de texto plano en un JSON estándar.

Los documentos pueden tener estructuras diferentes y no necesariamente deben tener todos los mismos campos ni su tipo. Por ejemplo, en el documento JSON de una estación meteorológica podemos incluir distintos campos como tipo y espesor si está nevando o l/m si está lloviendo. No necesariamente hay que incluir todos vacíos si es que ni nieva ni llueve.

En otros casos, los documentos JSON podrían incluir las coordenadas que delimitan una finca. Estas no necesariamente son 4 sino que dependerán de la forma de la finca.

```

1- {
2-   "listaEstacionsMeteo": [
3-     {
4-       "altitude": 21,
5-       "concello": "A CORUÑA",
6-       "estacion": "Coruña-Torre de Hércules",
7-       "idEstacion": 10157,
8-       "lat": 43.382763,
9-       "lon": -8.409202,
10-      "provincia": "A Coruña",
11-      "utmX": "547855.0",
12-      "utmY": "4803491.0"
13-    },
14-    {
15-      "altitude": 5,
16-      "concello": "A CORUÑA",
17-      "estacion": "Coruña-Dique",
18-      "idEstacion": 14000,
19-      "lat": 43.36506,
20-      "lon": -8.374706,
21-      "provincia": "A Coruña",
22-      "utmX": "550664.0",
23-      "utmY": "4801545.0"
24-    },
25-    {
26-      "altitude": 94,
27-      "concello": "ABEGONDO"
    }
  ]
}

```

Como se puede ver en la captura anterior de un JSON de MeteoGalicia, se parece mucho a un diccionario de Python. Ahí puedes ver como un documento JSON es un conjunto de pares clave valor. Además, también sabemos que los diccionarios pueden anidar documentos dentro de otros documentos.

Por tanto, MongoDB es la opción adecuada si los datos que manejamos son no estructurados o no tenemos aun clara la estructura necesaria y es probable que evolucione y cambie para adaptarse a las nuevas necesidades.

Mongo almacena los documentos en una carpeta concreta. Esta forma de almacenaje permite ampliar la base de datos horizontalmente añadiendo servidores y repartiendo la carga. Al distribuir la información en distintos nodos también es muy sencillo mantener réplicas de dos datos por si algún nodo falla.

La gran flexibilidad que permite mongo es también un problema al realizar las consultas. En bases de datos relacionales, las consultas son sencillas porque el esquema no cambia. En Mongo y, en general en todas las bases de datos NoSQL, las consultas son algo casi artesanal que requerirá algún ajuste a mano.

En esta guía veremos como aplicar comandos CRUD en mongo a través de comandos. Para ello será necesario tener correctamente configurado un servidor MongoDB y un cliente mongosh.

Documentos

MongoDB es una base de datos orientada a documentos. El documento es la unidad básica en este tipo de bases de datos.

```
{  
  "departamento":8,  
  "nombredepto":"Ventas",  
  "director": "Juan Rodríguez",  
  "empleados":[  
    {  
      "nombre":"Pedro",  
      "apellido":"Fernández"  
    }, {  
      "nombre":"Jacinto",  
      "apellido":"Benavente"  
    }  
  ]  
}
```

La visualización de un documento se realiza en JSON. Un formato de archivo basado en texto que se suele usar para intercambio de datos entre servidores y aplicaciones web. Su estructura es un conjunto de pares clave-valor separados por comas y encerradas todas ellas entre llaves. Las claves serán todas de tipo cadena. Los valores serán de tipo cadena, número, booleano, lista u otro documento anidado.

El almacenamiento interno de cada documento se hace con BSON, un formato equivalente a JSON pero en binario. Precisamente por ser binario con BSON se pueden usar otros tipos de datos como fechas, tipos numéricos más precisos u otros datos binarios para almacenar imágenes por ejemplo.

Ayuda de MongoDB

Siempre podremos consultar la web oficial de MongoDB shell en <https://www.mongodb.com/docs/mongodb-shell/reference/methods/> para obtener ayuda. Desde el propio Shell también podemos lanzar la función `help()`.

MongoDB es sensible a las mayúsculas así que debemos prestar atención al escribir los comandos.

Otra función muy útil es el uso del doble tabulador para completar comandos.

```
administrador@ubuntu:~$ mongo
Current MongoDB Log ID: 65931be9f1ab7e4c29842928
Connecting to: mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+2.1.0
Using MongoDB: 7.0.4
Using Mongosh: 2.1.0
Mongosh 2.1.1 is available for download: https://www.mongodb.com/try/download/shell
For mongosh info see: https://docs.mongodb.com/mongodb-shell/

-----
The server generated these startup warnings when booting
2024-01-01T18:49:22.756+00:00: Using the XFS filesystem is strongly recommended with the WiredTiger storage engine. See http://dochub.mongodb.org/core/prodnotes-filesystem
2024-01-01T18:49:23.602+00:00: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted
2024-01-01T18:49:23.602+00:00: vm.max_map_count is too low
-----

test>
test>
test> show
show databases      show dbs            show collections    show tables         show profile
show users          show roles          show log            show logs           show startupWarnings
show automationNotices show nonGenuineMongoDBCheck

test> show
```

En esta captura podemos ver qué pasa cuando empiezo a escribir un comando "show" y pulso doble tabulador para autocompletar. El resultado son los distintos parámetros que se pueden usar a continuación de "show".

Otras herramientas de MongoDB.

Al instalar MongoDB en nuestro sistema se instalarán también una serie de herramientas administrativas. Estas herramientas deben ser ejecutadas desde el sistema operativo como un ejecutable más. La instalación forma parte de una práctica anterior pero en la siguiente captura dejo un recuerdo del procedimiento muy resumido.

Install MongoDB Database Tools

Follow these steps to install **MongoDB 6.0 Community Edition on LTS (long-term support) releases of Ubuntu Linux**, using the **apt** package manager.

1. In the terminal, use the following command to import the public key used by the package management system:

```
wget -qO - https://www.mongodb.org/static/pgp/server-6.0.asc | sudo apt-key add -
```

2. Create a list file for MongoDB:

```
echo "deb [ arch=amd64,arm64 ] https://repo.mongodb.org/apt/ubuntu jammy/mongodb-org/6.0 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-6.0.list
```

3. Reload the local package database:

```
sudo apt-get update
```

4. Install the latest stable version of MongoDB Community Edition:

```
sudo apt-get install -y mongodb-org
```

Para copias de seguridad de bases de datos sencillas (sin sharding) podemos usar "mongodump" y "mongorestore". En nuestro caso usaremos "mongoexport" y "mongoimport" para gestionar las copias de seguridad de nuestras bases de datos. Estas últimas herramientas sí soportan bases de datos en sharding.

Con las herramientas "mongostat" y "mongotop" podremos diagnosticar y analizar las estadísticas que ofrece MongoDB y que nos aportan información para crear (y eliminar) índices que hagan que las consultas funcionen mucho más rápidas.

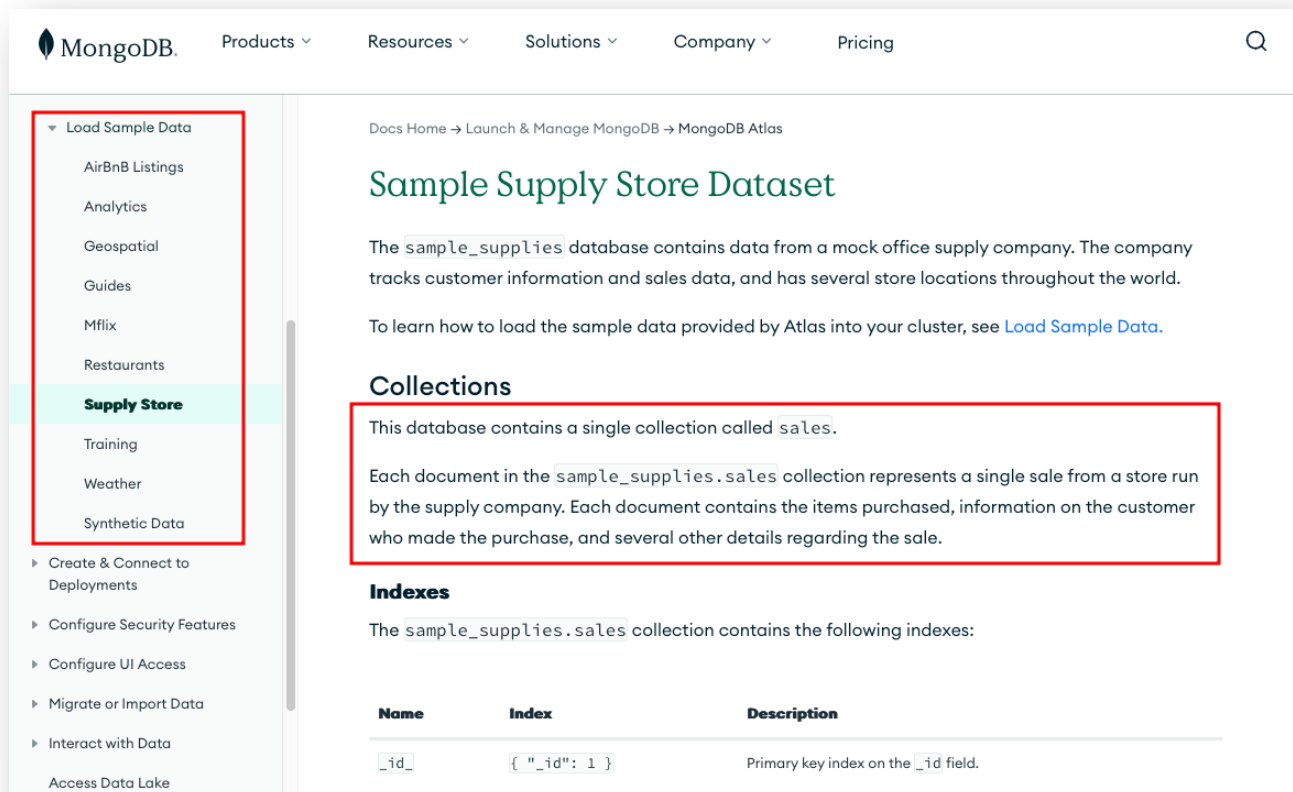
La herramienta "bsondump" nos permite depurar errores al leer un documento tal cual se almacena en disco y con ello comprobar si lo visto en JSON se corresponde.

Por último, la herramienta "mongofiles" nos permite interactuar con una base de datos MongoDB como si fuera un sistema de ficheros (GridFS). Una colección funciona como tabla de ficheros y otra colección actúa como repositorio de sectores donde se guardan trozos de archivo.

Descarga e instalación de bases de datos.

En la documentación de MongoDB se hace referencia a unas bases de datos de prueba con colecciones y documentos ficticios. Desde este repositorio de GitHub (<https://github.com/neelabalan/mongodb-sample-dataset>) podemos descargar en formato JSON las colecciones que nos interesen.

Para empezar descargaremos la base de datos "sample_supplies" con su única colección "sales".



Load Sample Data

- AirBnB Listings
- Analytics
- Geospatial
- Guides
- Mflix
- Restaurants
- Supply Store**
- Training
- Weather
- Synthetic Data

► Create & Connect to Deployments

► Configure Security Features

► Configure UI Access

► Migrate or Import Data

► Interact with Data

Access Data Lake

Docs Home → Launch & Manage MongoDB → MongoDB Atlas

Sample Supply Store Dataset

The `sample_supplies` database contains data from a mock office supply company. The company tracks customer information and sales data, and has several store locations throughout the world.

To learn how to load the sample data provided by Atlas into your cluster, see [Load Sample Data](#).

Collections

This database contains a single collection called `sales`.

Each document in the `sample_supplies.sales` collection represents a single sale from a store run by the supply company. Each document contains the items purchased, information on the customer who made the purchase, and several other details regarding the sale.

Indexes

The `sample_supplies.sales` collection contains the following indexes:

Name	Index	Description
<code>_id_</code>	<code>{ "_id": 1 }</code>	Primary key index on the <code>_id</code> field.

En la web de la documentación de MongoDB disponemos de la documentación suficiente para comprender el contenido de cada una de las bases de datos de prueba.

Para importar una base de datos debemos usar la herramienta "mongoimport" desde una terminal del sistema operativo.

```
administrador@ubuntu:~$ wget https://github.com/neelabalan/mongodb-sample-dataset/raw/main/sample_supplies/sales.json
--2024-01-01 19:52:02-- https://github.com/neelabalan/mongodb-sample-dataset/raw/main/sample_supplies/sales.json
Resolving github.com (github.com)... 140.82.121.4
Connecting to github.com (github.com)|140.82.121.4|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://raw.githubusercontent.com/neelabalan/mongodb-sample-dataset/main/sample_supplies/sales.json [following]
--2024-01-01 19:52:06-- https://raw.githubusercontent.com/neelabalan/mongodb-sample-dataset/main/sample_supplies/sales.json
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.109.133, 185.199.110.133, 185.199.111.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.109.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 4866111 (4.6M) [text/plain]
Saving to: 'sales.json'

sales.json           100%[=====] 4,64M  --.-KB/s  in 0,1s

2024-01-01 19:52:07 (41,1 MB/s) - 'sales.json' saved [4866111/4866111]

administrador@ubuntu:~$
```

```
wget https://github.com/neelabalan/mongodb-sample-dataset/raw/main/sample_supplies/sales.json
```

Con este comando descargamos el archivo JSON de la colección “sales” de la base de datos “sample_supplies”. Es el momento de importar este archivo JSON en nuestro servidor de MongoDB

```
administrador@ubuntu:~$ mongoimport -v --db supplies --collection sales --drop --host=127.0.0.1:27017 sales.json
2024-01-01T19:54:36.265+0000 using write concern: &{majority false 0}
2024-01-01T19:54:36.274+0000 filesize: 4866111 bytes
2024-01-01T19:54:36.279+0000 using fields:
2024-01-01T19:54:36.280+0000 connected to: mongodb://127.0.0.1:27017/
2024-01-01T19:54:36.280+0000 ns: supplies.sales
2024-01-01T19:54:36.281+0000 connected to node type: standalone
2024-01-01T19:54:36.282+0000 dropping: supplies.sales
2024-01-01T19:54:36.581+0000 5000 document(s) imported successfully. 0 document(s) failed to import.
administrador@ubuntu:~$
```

```
mongoimport -v --db supplies --collection sales --drop --host=127.0.0.1:27017 sales.json
```

Con este comando importamos el archivo JSON. Al leer la salida confirmamos que toda la información coincide con nuestros datos y, especialmente, la confirmación de que se han importado los 5000 documentos que componen esta colección.

Conexión a la base de datos

Desde un terminal conectamos al servidor MongoDB usando la ip del servidor y el puerto de escucha.

```
mongosh --host IP_SERVIDOR --port 27017
```

Si no hay problemas veremos la interfaz CLI de mongosh. Por defecto nos conectará a la base de datos “test”.

```
administrador@ubuntu:~$ mongosh
Current Mongosh Log ID: 65931a174a56f4d4049043cd
Connecting to: mongod://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+2.1.0
Using MongoDB: 7.0.4
Using Mongosh: 2.1.0
Mongosh 2.1.1 is available for download: https://www.mongodb.com/try/download/shell
For mongosh info see: https://docs.mongodb.com/mongodb-shell/

-----
The server generated these startup warnings when booting
2024-01-01T18:49:22.756+00:00: Using the XFS filesystem is strongly recommended with the WiredTiger storage engine. See http://dochub.mongodb.org/core/prodnot-tes-filesystem
2024-01-01T18:49:23.602+00:00: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted
2024-01-01T18:49:23.602+00:00: vm.max_map_count is too low
-----

test> _
```

En esta captura podemos confirmar que estamos conectados a nuestro servidor local. Por defecto, si lanzamos “mongosh” sin parámetros se intentará conectar al host 127.0.0.1 puerto 27017. Por defecto tenemos activa la base de datos llamada “test”, lo sabemos porque lo indica en el prompt.

Cambiar base de datos activa

En mongo no es necesario un comando para crear una base de datos. Simplemente pedimos usar una base de datos en concreto y, en caso de que no exista la creará en el mismo momento en que añadamos el primer documento a una colección.

```
administrador@ubuntu:~$ mongosh
Current Mongosh Log ID: 65799582334704de40a02fb6
Connecting to: mongod://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+2.1.0
Using MongoDB: 7.0.4
Using Mongosh: 2.1.0
For mongosh info see: https://docs.mongodb.com/mongodb-shell/

To help improve our products, anonymous usage data is collected and sent to MongoDB periodically (https://www.mongodb.com/legal/privacy-policy).
You can opt-out by running the disableTelemetry() command.

-----
The server generated these startup warnings when booting
2023-12-13T11:27:40.180+00:00: Using the XFS filesystem is strongly recommended with the WiredTiger storage engine. See http://dochub.mongodb.org/core/prodnot-tes-filesystem
2023-12-13T11:27:41.066+00:00: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted
2023-12-13T11:27:41.066+00:00: vm.max_map_count is too low
-----

test> use ciudades
switched to db ciudades
ciudades>
```

```
use mi_bd
```

Con este comando creo una nueva base de datos llamada “mi_bd” si no existiera ya, y la vuelvo activa. En general no se crea nada hasta el momento en que hay que hacer la primera escritura.

Insertar documentos en MongoDB

De la misma manera que las bases de datos, una colección se crea la primera vez que se guarda algo en ella.


```
mi_bd> db.ciudades.insertOne({
... nombre: "Vigo",
... poblacion: 292374,
... superficie: 109.06,
... hermanada: [ "Victoria de Durango", "Lorient", "Narsaq", "Oporto", "Buenos Aires", "Caracas", "Las Palmas", "Celaya", "Qingdao" ],
... autoridad: {
...   nombre: "Abel Caballero",
...   partido: "PSdeG-PSOE"
... }
... })
{
  acknowledged: true,
  insertedId: ObjectId('6579a6e569fe061120d547bf')
}
mi_bd>
```

```
db.ciudades.insertOne()
```

Mediante este comando inserto un único documento en la colección “ciudades” de la base de datos activa. El equivalente en SQL sería “INSERT INTO ciudades (nombre, población, superficie...) VALUES (“Vigo”, 292374, 109.05,...)”

Si nos fijamos en la captura vemos que a esta función le pasamos un diccionario entre llaves. Este diccionario contiene un conjunto de claves valor separados por comas.

La primera pareja es una clave “nombre”, dos puntos y una cadena de texto. Lo sabemos por las comillas.

Las dos siguientes parejas son números, el primero un entero y el segundo un decimal, Fijémonos en que el separador de los números decimales es un punto.

La cuarta pareja tiene la clave “hermanada”, dos puntos y una lista de cadenas de texto. Fijémonos en que se usan corchetes cuadrados para definir una lista, comillas para delimitar cadenas de texto y coma para separar cada elemento de la lista.

La quinta pareja tiene la clave “autoridad”, dos puntos y un nuevo documento o diccionario en su interior. Fijémonos en los corchetes curvos que indican inicio y fin de un documento. Este es un ejemplo de documento anidado.

Al pulsar intro se insertará ese documento en la colección. Fijémonos en que la salida del comando nos devuelve otro documento JSON con dos pares. Una es una confirmación de que la inserción se ha hecho bien, y en el segundo par se devuelve un identificador único para este documento que, además se insertará automáticamente dentro del documento.

```
db.ciudades.insertMany()
```

También disponemos de la función insertMany para insertar varios documentos a la vez.

Leer documentos en MongoDB

Para hacer búsquedas usamos la función find.

```
mi_bd> db.ciudades.find()
[
  {
    _id: ObjectId('6579db1db0fef9ea183d14d9'),
    nombre: 'Vigo',
    poblacion: 292374,
    superficie: 109.06,
    hermanada: [
      'Victoria de Durango',
      'Lorient',
      'Narsaq',
      'Oporto',
      'Buenos Aires',
      'Caracas',
      'Las Palmas',
      'Celaya',
      'Qingdao'
    ],
    autoridad: { nombre: 'Abel Caballero', partido: 'PSdeG-PSOE' }
  },
  {
    _id: ObjectId('6579dcf3b0fef9ea183d14da'),
    nombre: 'Santiago de Compostela',
    hermanada: [
      'Córdoba',
      'Santiago de Cali',
      'Temuco',
      'Buenos Aires',
      'Cáceres',
      'Qufu',
      'Pisa',
      'Le Puy-en-Velay',
      'Asís',
      'Coimbra'
    ],
    poblacion: 98179,
    autoridad: { nombre: 'Goretti Sanmartín Rei', partido: 'BNG' },
    superficie: 438.42
  }
]
mi_bd> 
```

```
db.ciudades.find()
```

Al ejecutar la función find sin argumentos nos devolverá una lista de diccionarios con todos los documentos que componen la colección desde donde es llamada, en este caso desde la colección "ciudades". El equivalente en SQL sería "SELECT * FROM ciudades".

```
mi_bd> db.ciudades.find( { "_id" : ObjectId("6579db1db0fef9ea183d14d9") } )
[
  {
    _id: ObjectId('6579db1db0fef9ea183d14d9'),
    nombre: 'Vigo',
    poblacion: 292374,
    superficie: 109.06,
    hermanada: [
      'Victoria de Durango',
      'Lorient',
      'Narsaq',
      'Oporto',
      'Buenos Aires',
      'Caracas',
      'Las Palmas',
      'Celaya',
      'Qingdao'
    ],
    autoridad: { nombre: 'Abel Caballero', partido: 'PSdeG-PSOE' }
  }
]
mi_bd>
```

```
db.ciudades.find( { "_id" : ObjectId("6579db1db0fef9ea183d14d9") } )
```

Para acceder a un documento en concreto podemos especificar algún argumento como por ejemplo el "_id" que se asigna automáticamente. Podemos ver como el parámetro que le hemos pasado a la función find es un documento un una sola clave y su valor. Fijémonos en que la respuesta es también una lista de un solo elemento. El equivalente en SQL sería "SELECT * FROM ciudades WHERE _id = xxxxx".

Podemos proyectar el resultado de la búsqueda para que solo nos muestre los campos que queremos

```
mi_bd> db.ciudades.find( { "_id" : ObjectId("6579db1db0fef9ea183d14d9") } , { nombre : true } )
[ { _id: ObjectId('6579db1db0fef9ea183d14d9'), nombre: 'Vigo' } ]
mi_bd>
```

```
db.ciudades.find( { "_id" : ObjectId("6579db1db0fef9ea183d14d9") } , { nombre : true } )
```

En este comando usamos la función "find" pero con dos documentos como parámetros. En el primero seguimos buscando el documento con el ID indicado, en el segundo parámetro indicamos con el valor "true" las claves que queremos proyectar.

El equivalente en SQL sería "SELECT _id, nombre FROM ciudades WHERE _id=xxxxx".

```
mi_bd> db.ciudades.find( { _id : ObjectId("6579db1db0fef9ea183d14d9") } , { hermanada : false } )
[
  {
    _id: ObjectId('6579db1db0fef9ea183d14d9'),
    nombre: 'Vigo',
    poblacion: 292374,
    superficie: 109.06,
    autoridad: { nombre: 'Abel Caballero', partido: 'PSdeG-PSOE' }
  }
]
mi_bd> █
```

```
db.ciudades.find( { _id : ObjectId("6579db1db0fef9ea183d14d9") } , { hermanada : false } )
```

En este comando hacemos justo lo contrario. Seguimos buscando un documento concreto pero en esta ocasión no queremos ver el valor de la clave "hermanada".

En MongoDB también contamos con el método "findOne()" que funciona igual que "find()" pero solo devuelve un resultado.

Por ahora es suficiente con estas consultas. En otra guía desarrollaremos más las consultas pero para quien quiera adelantar sería bueno echar un ojo a las expresiones regulares.

Agregados de documentos en MongoDB

Los agregados son un tipo de consulta en la que no se busca obtener un listado de documentos y sí un dato resumen. Un ejemplo típico es el siguiente comando.

```
db.ciudades.find().count()
```

El resultado de este comando es solamente un número que indica cuantos documentos hay en la colección ciudades.

```
db.ciudades.find().limit(2)
```

El resultado de este comando son los dos primeros documentos de la colección "ciudades".

La función más potente en los agregados es la función aggregate() a la que podemos pasar 3 parámetros equivalentes al WHERE, GROUP BY y ORDER BY de una sentencia SQL.

La función aggregate está compuesta por etapas, por "stages", que van agrupando, ordenando, limitando y otras muchas operaciones como modificar, etc.

Actualizar documentos en MongoDB

Para actualizar documentos en MongoDB debemos tener en cuenta que podemos actualizar todo el documento o solo alguno de sus valores.

```
mi_bd> db.ciudades.find( { "_id" : ObjectId("6579db1db0fef9ea183d14d9") } )
[
  {
    _id: ObjectId('6579db1db0fef9ea183d14d9'),
    nombre: 'Vigo',
    poblacion: 292374,
    superficie: 109.06,
    hermanada: [
      'Victoria de Durango',
      'Lorient',
      'Narsaq',
      'Oporto',
      'Buenos Aires',
      'Caracas',
      'Las Palmas',
      'Celaya',
      'Qingdao'
    ],
    autoridad: { nombre: 'Abel Caballero', partido: 'PSdeG-PSOE' }
  }
]
mi_bd> db.ciudades.updateOne( { "_id" : ObjectId("6579db1db0fef9ea183d14d9") }, { $set : {poblacion:300000}} )
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
mi_bd> db.ciudades.find( { "_id" : ObjectId("6579db1db0fef9ea183d14d9") } )
[
  {
    _id: ObjectId('6579db1db0fef9ea183d14d9'),
    nombre: 'Vigo',
    poblacion: 300000,
    superficie: 109.06,
    hermanada: [
      'Victoria de Durango',
      'Lorient',
      'Narsaq',
      'Oporto',
      'Buenos Aires',
      'Caracas',
      'Las Palmas',
      'Celaya',
      'Qingdao'
    ],
    autoridad: { nombre: 'Abel Caballero', partido: 'PSdeG-PSOE' }
  }
]
mi_bd> 
```

```
db.ciudades.updateOne( { _id : ObjectId("6579db1db0fef9ea183d14d9") } , { $set: {poblacion : 300000} } )
```

Con este comando modificamos el documento original y cambiamos el valor de su población. Fijémonos en el "\$set" que indica que solo queremos modificar el valor de esa clave. Si no ponemos el "\$set" estaremos cambiando todo el documento por otro en el que solo está ese clave valor.

El equivalente en SQL sería "UPDATE ciudades SET población = 300000 WHERE _id = xxxx"

Existe también la clave \$unset que sirve para eliminar el campo.

Existe el comando "updateMany()" que funciona exactamente igual pero que afecta a ninguno, uno o varios documentos.

Borrar documentos en MongoDB

El borrado funciona exactamente igual que la consulta sin más que sustituir la función find por un deleteOne.

```
db.ciudades.deleteOne( { nombre : "Santiago de Compostela" } )
```

Aquí no hay confirmaciones de ningún tipo, lo borra y punto. Lo único que obtendremos es un JSON de resultado en el que nos indica que se ha ejecutado el comando y la cantidad de eliminaciones que ha hecho.

El equivalente en SQL sería "DELETE FROM ciudades WHERE nombre="Santiago de Compostela""

Además de "deleteOne()" también disponemos de "deleteMany()" que permite eliminar varios documentos en el mismo comando.